Interviewing Tips:

**What is Bombay Play looking for?**

*"We are not simply looking for engineers to solve the problems they already know the answers to; we are interested in engineers who can work out the answers to questions they have not come across before."*

**Interviewers will be looking at the approach to questions as much as the answer...**

* Does the candidate listen carefully and comprehend the question?
* Are the correct questions asked before proceeding? (important!)
* Is brute force used to solve a problem? (not good!)
* Are things assumed without first checking? (not good!)
* Are hints heard and heeded?
* Is the candidate slow to comprehend / solve problems? (not good!)
* Does the candidate enjoy finding multiple solutions before choosing the best one?
* Are new ideas and methods of tackling a problem sought?
* Is the candidate inventive and flexible in their solutions and open to new ideas?
* Can questioning move up to more complex problem solving?

**Tips on how to prepare for an interview at Bombay Play:**

Bombay Play is keen to see really high quality, efficient, clear code without typing mistakes. Because all engineers (at every level) collaborate throughout the Bombay Play code base, with an efficient code review process, it's essential that every engineer works at the same high standard.

The interviewers are keen to assess your ability to develop original software in a fairly short period of time. Interview topics may cover anything on your CV (especially if you have stated that you are an expert), building and developing complex algorithms and analysing their performance characteristics, logic problems, systems design and fundamental computer science principles ‐ **hash tables, stacks, arrays, data‐structures, object oriented programming skills, algorithms, etc. and how they can be used in your solution**.

**This is the important bit:**

For instance, he/she may set a deliberately ambiguous real‐world problem and ask you to find solutions to it. You'll need to interpret the coding knowledge that you have for that particular situation. They are looking for process of thought, creative solutions and being able to work out more than one way to solve a problem and talk through your rationale for choosing a certain way to approach solving the problem. So, you could perhaps recommend an algorithm, code up a solution using that algorithm, analyze the runtime of your code and then optimize your solution. Engineers at Bombay Play each handle all aspects of system architecture, design, coding and testing, developing quality modules without job hierarchy. Therefore, computer science fundamentals are pre‐requisite for all engineering roles at Bombay Play, regardless of seniority.

**Ask more questions!**

At Bombay Play, we believe in collaboration and sharing ideas. Most importantly, you'll need more information from the interviewer to analyze & answer the question to its full extent.

It’s great to question your interviewer!

* When asked to provide a solution, first define and framework the problem as you see it.
* If you don't understand ‐ ask for help or clarification.
* If you need to assume something ‐ verbally check it's a correct assumption!
* Describe how you want to tackle solving each part of the question.
* Always let your interviewer know what you are thinking as he/she will be as interested in your process of thought as your solution. Also, if you're stuck, they may provide hints if they know what you're doing.
* Finally, listen ‐ don't miss a hint if your interviewer is trying to assist you!

At the end of the interview, most interviewers will ask you if you have any questions about the company, work environment, their experience, etc. It's clever to have some pre‐prepared for each interview, but don't worry too much if your mind goes blank. Note - If you have questions about the interview process, remuneration or your performance, please direct these to your recruiter.

**Technical Preparation tips!**

The main areas software engineers should prepare to succeed at interview at Bombay Play:

**Algorithm Complexity:** You need to know Big‐O. If you struggle with basic big‐O complexity analysis, then you are almost guaranteed not to get hired.

**Sorting:** Know how to sort. Don't do bubble-sort. You should know the details of at least one n\*log(n) sorting algorithm, preferably two (say, quicksort and merge sort). Merge sort can be highly useful in situations where quicksort is impractical, so take a look at it.

**Hashtables:** Arguably the single most important data structure known to mankind. You absolutely should know how they work. Be able to implement one using only arrays in your favorite language, in about the space of one interview.

**Trees:** Know about trees; basic tree construction, traversal and manipulation algorithms. Familiarise yourself with binary trees, n‐ary trees, and trie‐trees. Be familiar with at least one type of balanced binary tree, whether it's a red/black tree, a splay tree or an AVL tree, and know how it's implemented. Understand tree traversal algorithms: BFS and DFS, and know the difference between inorder, postorder and preorder.

**Graphs:** Graphs are really important. There are 3 basic ways to represent a graph in memory (objects and pointers, matrix, and adjacency list); familiarize yourself with each representation and its pros & cons. You should know the basic graph traversal algorithms: breadth‐first search and depth‐first search. Know their computational complexity, their tradeoffs, and how to implement them in real code. If you get a chance, try to study up on fancier algorithms, such as Dijkstra and A\*.

**Other data structures:** You should study up on as many other data structures and algorithms as possible. You should especially know about the most famous classes of NP‐complete problems, such as traveling salesman and the knapsack problem, and be able to recognize them when an interviewer asks you them in disguise. Find out what NPcomplete means.

**Mathematics:** Some interviewers ask basic discrete math questions. This is more prevalent at Bombay Play than at other companies because we as a gaming company are surrounded by counting problems, probability problems, and other Discrete Math 101 situations. Spend some time before the interview refreshing your memory on (or teaching yourself) the essentials of combinatorics and probability. You should be familiar with n‐choose‐k problems and their ilk – the more the better.

**Systems Design:** You need to know powers of 2, and be good and back-of-the-enveloper calculations e.g. to estimate the required number of machines for a given design. Know some famous games, and think about how you would design the back-end (of the front-end). System design questions are a test of your problem solving. Sample topics include: **features sets, interfaces, class hierarchies, distributed systems, designing a system under certain constraints, simplicity and robustness, trade offs.**

**Operating Systems:** Know about processes, threads and concurrency issues. Know about locks and mutexes and semaphores and monitors and how they work. Know about deadlock and livelock and how to avoid them. Know what resources a process needs, and a thread needs, and how context switching works, and how it's initiated by the operating system and underlying hardware. Know a little about scheduling. The world is rapidly moving towards multi‐core, so know the fundamentals of "modern" concurrency constructs.

**Coding:** This is most important. You should know at least one programming language really well. You will be expected to write some code in all of your interviews. You will be expected to know a fair amount of detail about your favorite programming language and also be expected to produce code on a whiteboard/whitepaper. Your interviewer will be looking for logically and syntactically correct (compilable) code - **NOT PSEUDOCODE**, so perhaps to practice, try writing code on paper, then put it through a compiler to make sure that your code compiles (and if it doesn't, then practice some more).

To practice for your interview you may want to;

Visit the website [TopCoder.com](http://topcoder.com) ‐ if you launch the "Arena" widget and then go to the practice rooms where you can play with the problems in the first/second division as a warm up.

**OR**

This is a useful site for practicing coding which is not competitive and doesn't require that you download additional apps: <http://projecteuler.net/>

**Final tips!**

* **Start preparing early.** Your interview will be mainly technical, so give yourself at least a week or 2 weeks to review and hone your coding, algorithmic and problem solving skills.
* Get your algorithms straight (they may comprise up to a third of your interview). Visit:

[List of Algorithm general topics](http://en.wikipedia.org/wiki/List_of_algorithm_general_topics)

[List of Algorithms](http://en.wikipedia.org/wiki/List_of_algorithms)

[List of Data Structures](http://en.wikipedia.org/wiki/List_of_data_structures)

* **Write out all the algorithms yourself** from start to finish and make sure they're working.
* **Read up on all the latest news about your preferred programming language(s)**. If you say you're an expert in Java, you'll be expected to know what's new in the field.
* **Get in a creative 'problem solving' mood**. Spend some time everyday doing Sudoku, word puzzles‐‐anything to exercise that part of your brain.
* **Most of your interviews will be technical**‐‐but be ready to provide info about your past experience and major accomplishments. Make sure you can explain the architecture of previous projects.